On 2-Round Secure Multiparty Computation

Rosario Gennaro[1], Yuval Ishai[2⋆], Eyal Kushilevitz[3⋆⋆], and Tal Rabin[1]

[1] IBM T.J. Watson Research Center. E-mail: {rosario,talr}@watson.ibm.com.
[2] Princeton University. E-mail: yishai@cs.princeton.edu.
[3] Technion, Israel. E-mail: eyalk@cs.technion.ac.il.

Abstract. Substantial efforts have been spent on characterizing the round complexity of various cryptographic tasks. In this work we study the round complexity of *secure multiparty computation* in the presence of an *active* (Byzantine) adversary, assuming the availability of secure point-to-point channels and a broadcast primitive. It was recently shown that in this setting *three* rounds are sufficient for arbitrary secure computation tasks, with a linear security threshold, and two rounds are sufficient for certain nontrivial tasks. This leaves open the question whether *every* function can be securely computed in two rounds.
We show that the answer to this question is "no": even some very simple functions do not admit secure 2-round protocols (independently of their communication and time complexity) and thus 3 is the exact round complexity of general secure multiparty computation. Yet, we also present some positive results by identifying a useful class of functions which can be securely computed in two rounds. Our results apply both to the information-theoretic and to the computational notions of security.

Key Words: Secure multiparty computation, round complexity, lower bounds.

1 Introduction

The race for improving the *round complexity* of cryptographic protocols appears, quite miraculously, to never lose its steam.[1] In this work we study the round complexity of *secure multiparty computation*. Following the initial plausibility results in this area [32, 21, 5, 10], considerable efforts have been spent on obtaining round-efficient protocols [1, 4, 3, 13, 22, 31, 2, 8, 23, 17, 26, 11, 24]. In the multiparty setting, it was recently shown in [17] that every function can be securely computed in three rounds (tolerating a constant fraction of *malicious* players)[2] and that for certain nontrivial tasks two rounds suffice. This

⋆ Most of this work was done while the author was at AT&T Labs—Research.

⋆⋆ Most of this work was done while the author was at IBM T.J. Watson Research Center.

[1] It is somewhat reassuring to note in this context that the round complexity is restricted to be a *positive* integer.

[2] This is possible either with unconditional security, efficiently in the branching program size of the function being computed, or with computational security, efficiently in its circuit size.

naturally raises the question whether *every* function can be securely computed in two rounds. In the current work we focus on this question, examining the capabilities and limitations of 2-round protocols.

The Model. We consider a system of n players, who interact in synchronous rounds via authenticated secure point-to-point channels and a broadcast medium[3]. Interfering with the interaction is an *active* (Byzantine) adversary, who may corrupt up to t players (where t is referred to as the *security threshold*), learn their internal data, and arbitrarily modify their behavior. By default, we make the standard assumption that the adversary has a *rushing* capability, namely in each round it may learn the messages sent at this round by uncorrupted players to the corrupted players before sending its own messages. This is the most commonly used model in the general secure multiparty computation literature (e.g., [21, 5, 10, 30, 28, 12]), and in particular it is the standard model assumed in the context of *constant-round* secure multiparty computation (e.g., [1, 4, 3, 2, 23, 17]). We will also address the situation in the *fully synchronous* setting, where the messages of each round are guaranteed to be simultaneous. As for other aspects of the model, such as perfect vs. computational security, and adaptive vs. non-adaptive adversary, they can be set appropriately so as to achieve the strongest statements for both our positive and negative results. Section 2 includes a more detailed description of the model and the standard definition of security in this model.

1.1 Our Results

We obtain both positive and negative results, resolving the main qualitative questions concerning the possibility of 2-round secure multiparty computation. Our main results are outlined below.

Positive results. We show that any function whose outputs are determined by the input of a *single* player can be securely computed in two rounds with a linear security threshold, efficiently in its circuit size. In contrast to its appearance, this function class is neither trivial nor useless. In particular, this result generalizes the 2-round protocols for verifiable secret-sharing and secure multicast from [17], implies 2-round distributed zero-knowledge protocols for NP [16], and also has other applications that we discuss. In addition, we observe that there are also functions outside this class which admit 2-round protocols.

Negative results. Our main conclusion is that 3 rounds are necessary for general secure multiparty computation in any "standard" model (i.e., without public-key infrastructure or preprocessing and with full fairness requirement; see below). Specifically, if $t \geq 2$ (i.e., at least two players may be corrupted), then

[3] Broadcast allows each player to send an identical message to *all* players, without allowing it to violate the consistency requirement. This primitive can be simulated using secure point-to-point channels via a Byzantine Agreement protocol [29, 25, 7, 14]; however the cost of such a simulation would exceed the round complexity we consider here. From a more practical point of view, a broadcast medium may be implemented either physically or via semi-trusted external parties, such as Internet bulletin boards.

even some very simple functions cannot be securely computed in two rounds, regardless of the number of players, n, and the protocol's communication and time complexity. We consider the simple special cases of XOR_2^n and AND_2^n (exclusive-or and conjunction of two input bits, where all n players should learn the output) and show that both functions do not admit 2-round protocols. Interestingly, these two cases turn out to be quite different from each other, and each proof represents a distinct type of security concern. Indeed, the impossibility result in the former case is inherently linked to the adversary's rushing capability, as XOR_2^n (and more generally, any *linear* function) admits 2-round protocols in the fully synchronous model. In contrast, the second negative result (for the function AND_2^n) applies also to the fully synchronous model. Naturally, the above two special cases generalize to larger classes of functions with "similar" properties. Combining the above negative results with the 3-round upper bound of [17], we have that 3 is the *exact* round complexity of general secure multiparty computation with a linear security threshold.

Extension to more liberal models. We note that the above negative results can be extended to the common random string model or, more generally, to a setting where a public string from an arbitrary trusted distribution is given to *all* players. However, the results do not apply if we allow either a preprocessing stage or distribution of correlated random resources by a trusted party prior to the computation. In fact, in these models two rounds are sufficient for securely computing every function with a linear security threshold. For instance, [18] shows that two rounds are sufficient for achieving *independence* (i.e., a simultaneous broadcast among n players) given a public-key infrastructure. It follows from our results that some underlying infrastructure is indeed necessary, as otherwise the corresponding functionality is impossible to compute securely in two rounds. Our negative results also rely on the *fairness* requirement of secure computation, and do not apply if the adversary is allowed to "abort" the computation after learning its output.

1.2 Related Work

Most relevant to the current work are the works on the round complexity of secure multiparty computation, cited above. Among those, the only work to prove *lower bounds* on the round complexity is [17], where it is proved that *perfect* VSS and secure multicast with *optimal resilience* ($t < n/3$) require 3 rounds. However, settling for a slightly smaller security threshold ($t < n/4$), these tasks require only two rounds. In contrast, our negative results for 2 rounds apply even when the number of players n is arbitrarily larger than t, and even for the relaxed notions of statistical and computational security. We stress though that our negative results do not apply to more liberal settings of secure computation. For instance, if the adversary is *passive*, then two rounds are sufficient for computing any function with $t < n/3$; this can be achieved either with computational security, efficiently in the circuit size [4], or with perfect security, efficiently in the branching program size [24]. In the *two-party* case, 2-round secure computation is possible either against a passive adversary in the standard model [32, 31], or against an active adversary in the common reference string model, assuming that only one player has an output [8].

The round complexity of *zero-knowledge* protocols has been extensively studied in various settings (e.g., see [20]). However, this line of work is very different from ours both in the type of task being considered (zero-knowledge vs. general secure computation) and, more importantly, in the setting. Indeed, our multiparty setting is more liberal in the sense that it only requires security against limited collusions of players, and in particular allows to assume that a strict majority of the players remain uncorrupted.

Various papers deal with the round complexity of implementing *Byzantine Agreement* and broadcast using only point-to-point channels (e.g., see [29, 14, 27]). While these problems can be viewed as secure computation tasks, they are trivialized in our model since we assume broadcast as a primitive.

Finally, the round complexity of *collective coin-flipping* (which may also be viewed as a secure computation task) has been discussed in the *full information model* [6]. This model is dual to the Byzantine Agreement one: it allows *only* broadcast and no secure point-to-point communication. Similarly to our default model, the adversary is allowed rushing. We note that the availability of secure point-to-point channels in our model makes the coin-flipping task more feasible, and thus negative results from the relevant literature do not apply in our context.

Organization. In Section 2 we present the model and definitions. Section 3 includes our positive results, followed by the lower bounds in Section 4.

2 Model and Definitions

In this section we outline the definition of secure computation, following Canetti's definition approach [9], and highlight some details that are important for our purposes. The following version of the definition is somewhat simplified. In particular, this simplified version considers a protocol as a stand-alone application and does not support any kind of composition; however, our positive results (and obviously the negative results) hold for stronger versions of the definition as well. We refer the reader to [9, 19] for more complete definitions.

Communication model. We consider a network of n processors, denoted $P_1, \ldots, P_n$ and referred to as *players*. Each pair of players is connected via a private, authenticated point-to-point channel. In addition, all players share a common *broadcast* channel, which allows a player to send an identical message to *all* other players. In some sense, the broadcast channel can be viewed as a medium which "commits" the player to a specific value.

Function. A secure computation task is defined by some *n-party function* $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, specifying the desired mapping from the players' inputs to their final outputs. While in certain interesting cases the players will have to reach an agreement on a joint output, the definition allows for each player to compute its own output. When referring to a single-output function, it is assumed by default that all players output its value. One may also consider *randomized* functions, which take an additional random input; however, in this work we focus by default on the deterministic case.

Protocol. Initially, each player P_i holds an input x_i, a random input r_i, and a common *security parameter* k. The players are restricted to (expected) polynomial time in k. The protocol proceeds in *rounds*, where in each round each player P_i may send a "private" message to each player P_j (including itself) and broadcast a "public" message, to be received by all players. The messages P_i sends in each round may depend on all its inputs (x_i, r_i and k) and the messages it received in previous rounds. From now on, we assume without loss of generality that each P_i sends x_i, r_i, k to itself in the first round, so that the messages it sends in each subsequent round may be *determined* from the messages received in previous rounds. We assume that the protocol terminates after a fixed number of rounds, and each player locally outputs some function of the messages it received.

Adversary. We consider an *active* t*-adversary* $\mathcal{A}$, where the parameter t is referred to as the *security threshold*. The adversary is an efficient interactive algorithm,[4] which is initially given the security parameter k and a random input r. Based on these, it may choose a set T of *at most* t players to corrupt.[5] The adversary then starts interacting with a protocol (either a "real" protocol as above, or an *ideal-process* protocol to be defined below), where it takes control of all players in T. In particular, it can read their inputs, random inputs, and received messages, and it can fully control the messages they send. (In the weaker setting of *passive* security, the adversary cannot modify the corrupted players' behavior, but only read their information.) We assume by default that the adversary has a *rushing* capability: at any round it can first wait to hear all messages sent by uncorrupted players to players in T, and use these to determine its own messages. However, we also consider the *fully synchronous* model, in which the messages sent by the adversary in each round are independent of the messages sent by uncorrupted players in the same round. Finally, upon the protocol's termination, $\mathcal{A}$ outputs some function of its entire view.

Security. Informally, a protocol computing f is said to be t-secure if whatever a t-adversary can "achieve" by attacking the protocol, it could have also achieved (by corrupting the same set of players) in an ideal process in which f is evaluated using a trusted party. To formalize this definition, we have to define what "achieve" means and what the ideal process is. The *ideal process* for evaluating the function f is a protocol π_f involving the n players and an additional, incorruptible, trusted party TP. The protocol proceeds as follows: (1) each P_i sends its input x_i to TP; (2) TP computes f on the inputs (using its own random input in the randomized case), and sends to each player its corresponding output. Note that when an adversary corrupts the ideal process, it can pick the inputs sent by players in T to TP (possibly, based on their original inputs) and then output an

[4] It is usually assumed that the adversary is given an "advice" string a, or is alternatively modeled by a nonuniform algorithm. In fact, the proofs of our negative results are formulated in this nonuniform setting, but can be modified to apply in the uniform one as well.

[5] This corresponds to the *non-adaptive* security model; however, all our results apply to the stronger adaptive model as well.

arbitrary function of all its view (including the outputs it received from TP). To formally define security, we capture what the adversary "achieves" by a random variable concatenating the adversary's output together with the *outputs* and the *identities* of the uncorrupted players. For a protocol π, adversary $\mathcal{A}$, input vector x, and security parameter k, let $exec_{\pi,A}(k,x)$ denote the above random variable, where the randomness is over the random inputs of the uncorrupted players, the trusted party (if f is randomized), and the adversary. The security of a protocol Π (also referred to as a *real-life* protocol) is defined by comparing the *exec* variable of the protocol Π to that of the ideal process π_f. Formally:

Definition 1. *We say that a protocol Π* t-securely computes f *if, for any (real-life) t-adversary $\mathcal{A}$, there exists (an ideal-process) t-adversary $\mathcal{A}'$ such that the distribution ensembles $exec_{\Pi,\mathcal{A}}(k,x)$ and $exec_{\pi_f,\mathcal{A}'}(k,x)$ are* indistinguishable. *The security is referred to as perfect, statistical, or computational according to the notion of indistinguishability being achieved. For instance, in the computational case it is required that for any family of polynomial-size circuits $\{C_k\}$ there exists some* negligible *function* neg*, such that for any x,*

$$|C_k(exec_{\Pi,\mathcal{A}}(k,x)) - C_k(exec_{\pi_f,\mathcal{A}'}(k,x))| \leq \mathsf{neg}(k).$$

An equivalent form of Definition 1 quantifies over all *input distributions* X rather than specific input vectors x, and gives X as an additional input to the distinguisher C_k. This equivalent form is convenient for proving our negative results.

Intuitive discussion. Definition 1 asserts that for any *real-life* t-adversary $\mathcal{A}$ attacking the real protocol there is an *ideal-process* t-adversary $\mathcal{A}'$ which can "achieve" in the ideal process as much as $\mathcal{A}$ does in the real life. The latter means that the output produced by $\mathcal{A}'$ together with the inputs and outputs of uncorrupted players in the ideal process is indistinguishable from the output (wlog, the entire view) of $\mathcal{A}$ concatenated with the inputs and outputs of uncorrupted players in the real protocol. This concatenation captures both *privacy* and *correctness* requirements. On the one hand, it guarantees that the view of $\mathcal{A}$ does not allow it to gain more information about inputs and outputs of uncorrupted players than is possible in the ideal process and, on the other hand, it ensures that the inputs and outputs of the uncorrupted players in the real protocol be consistent with some correct computation of f in the ideal process.

Additional intuition regarding the definition, including our general paradigm for proving *negative* results, is given in Section 4.

3 Positive Results

Which functions are the easiest to compute securely? In this section we obtain a 2-round protocol for every function whose outputs are determined by the input of a *single* player. We stress that this class of secure computation tasks is nontrivial, and in fact it can be used to implement some important tasks such as VSS and distributed zero-knowledge. To see that this class is nontrivial, note that in the multi-output case the protocol has to ensure that all local outputs be consistent with the same input, and at the same time must hide the players' outputs from each other. Moreover, even in the single-output case (i.e., where the same output is learned by all players) it is not enough to let the player holding the input compute and broadcast the global output; indeed, in this naive protocol the players may not be able to *efficiently* verify that the broadcasted value is consistent with some valid input.

We now show that every function in the above class can be securely computed in 2 rounds with perfect security and a linear security threshold, efficiently in its circuit size. To prove our claim, we reduce the task of securely computing such a function to that of securely computing a related vector of degree-2 polynomials, and in return show how to compute such a vector in 2 rounds. We start by describing the latter.

Lemma 1. *Let $p = (p_1, \ldots, p_s)$ be a vector of degree-2 multivariate polynomials in the inputs $x = (x_1, \ldots, x_m)$ over a finite field F, where $|F| > n$.*[6] *Moreover, suppose that P_1 holds the entire input vector x and that each player gets some specified subset of the outputs. Then, $p(x)$ admits a perfectly secure 2-round protocol with a linear security threshold in which the communication and time complexity are linear in $s + m$ and polynomial in the number of players.*

Proof sketch: The protocol proceeds similarly to the 2-round VSS protocol from [17]. Here we outline a somewhat simplified version which does not achieve an optimal security threshold, but suffices for our purposes. For simplicity, assume that $s = 1$; the general case is handled by parallel repetition. In the first round, P_1 uses the bivariate polynomial secret-sharing of [5] to share each of its inputs; that is, it chooses a random bivariate polynomial $F^l(y, z)$ over F of degree at most t in each variable under the condition that $F^l(0, 0) = x_l$ for $l = 1, 2, \ldots, m$. It sends to player P_i the polynomials $f_i^l(y) = F^l(y, i)$ and $g_i^l(z) = F^l(i, z)$. In parallel, each pair of players privately exchange random pads. In the second round, each player P_i lets its primary share of x_l be $s_i^l = f_i^l(0)$ (note that if P_1 is honest then the points (i, s_i^l) lie on a degree-t polynomial) and sends, to each player who should receive the output, the value $p(s_i^1, \ldots, s_i^m)$, i.e. its share of the output $p(x)$.[7] In parallel, each player P_j broadcasts the value of each *secondary* share $f_i^l(j)$ and $g_i^l(j)$ masked with the pad exchanged with P_j in Round 1. These broadcasts induce an *inconsistency graph*, each edge of which represents a conflict between secondary shares of different players that were supposed to be equal (if P_1 and the two relevant players were honest).

We now describe how each player reconstructs the value of p from the n output shares it received and from the (public) inconsistency graph. Suppose that $n > 6t$. The players run a deterministic 2-approximation algorithm for vertex cover on the inconsistency graph [15]. If it returns a vertex cover of size$> 2t$

[6] This assumption on the size of F can be eliminated by the use of extension fields.

[7] To guarantee privacy, these values have to be randomized so that they lie on a *random* degree-2t polynomial with $p(x)$ as its free coefficient. We ignore this detail, since it complicates the presentation and is addressed in a standard way by letting P_1 share additional random values.

(implying that there is no vertex cover of size t), then it is clear that P_1 is dishonest, and the output is taken to be $p(0)$. Otherwise, let I be the complement of the vertex cover (which is an independent set in the graph). Note that $|I| > 4t$, and so the players in I contain at least $3t+1$ uncorrupted players whose input shares were all consistent with some input vector $x' = (x'_1, \ldots, x'_m)$, and thus their output shares lie on a degree-2t polynomial with free coefficient $p(x')$. The output value is computed from the $|I|$ output shares of the players in I by applying a Reed-Solomon error correction procedure to find the "nearest" degree-$2t$ polynomial, and taking its free coefficient. Note that if P_1 is uncorrupted then, since the distance of the relevant code is greater than $2t$, the correct output will be computed. Conversely, if I indeed contains more than $4t$ players, then the output will be consistent with the value of p on some input x' defined by the (consistent) shares of the uncorrupted players in I. □

Theorem 1. *Suppose that f is a deterministic function whose inputs are all held by a single player. Then, f admits a perfectly secure 2-round protocol with a linear security threshold, computing f efficiently in its circuit size.*

Proof. We prove the theorem for the case of a single-output function represented by a boolean circuit; a proof for the general case proceeds similarly. We reduce the secure computation of f to the secure computation of degree-2 polynomials. Suppose that C is a boolean circuit computing f, and let F be a finite field where $|F| > n$. We construct a vector p of degree-2 polynomials over F. The input variables of p are of three types: (1) variables x, such that x_i corresponds to the i-th input of f; (2) variables y, such that y_i corresponds to the i-th intermediate wire in C (i.e., excluding input and output wires); (3) variables z, such that z_i corresponds to the i-th output wire of C. The vector $p(x, y, z)$ will serve to verify that the input values x are valid (i.e., each of these variables is assigned either 0 or 1), and that the wire labels y, z are consistent with the gates of C and the inputs x. Specifically, it should hold that $p(x, y, z)$ is the zero vector if all the above consistency requirements are met, and otherwise it contains at least one nonzero entry. Note that each atomic consistency requirement can be verified by a single degree-2 polynomial. For instance, the validity of an input value x_i can be verified by the polynomial $x_i(1 - x_i)$, and the consistency of an internal NAND gate having input wires i, j and output wire k can be verified by $1 - y_i y_j - y_k$. Hence, the total length of p is proportional to the size of C.

Now, let $p'(x, y, z) = p(x, y, z) \circ z$ (where $\circ$ denotes concatenation). Given a secure protocol Π for computing $p'(x, y, z)$, a secure protocol for $C(x)$ proceeds as follows:

- On input $x \in \{0,1\}^m$, player P_1 computes the wire labels y, z, and invokes Π on inputs x, y, z. Let $v \circ z$ denote the output of Π (by its security, the same output must be obtained by all uncorrupted players).
- Each player computes its output as follows: If $v = 0$ output z, otherwise output $C(0)$.

The correctness of this reduction can be sketched as follows. If P_1 is honest, the correct output $z = C(x)$ will clearly be obtained (even in the presence of an active t-adversary), and no additional information about x will be revealed. Conversely, for either of the two possibilities for obtaining the output, it must be consistent with some input x. □

As a corollary of Theorem 1, it is possible to obtain 2-round distributed zero-knowledge protocols for NP [16]. Indeed, if R is a polynomial-time predicate defining an NP-language, P_1 can prove that it knows a witness w such that $R(x, w) = 1$ by invoking a secure protocol for the function $f(x, y) = x \circ R(x, y)$ (substituting w for y). Theorem 1 can also be applied for obtaining a wide array of "certified secret-distribution" schemes, generalizing the VSS primitive. For instance, let $D(s, r)$ be a (t, n)-secret-sharing scheme (such that $D_i(s, r)$ is the share of the secret s held by P_i), and let $R(s)$ be an efficient predicate testing whether s satisfies some validity condition. Define a function f whose inputs s, r are held by P_1, and such that P_i's output is $(D_i(s, r), R(s))$. Then, the secure computation of f allows P_1 to securely distribute his secret s among n players ensuring consistency of the shares with some *valid* secret s, which can at a later stage be reconstructed even in the presence of faulty players. Note that if P_1 fails to pick the input r at random, then at most the *secrecy* of the secret s is compromised (which anyway cannot be avoided) but not its validity.

We end this section by noting that Theorem 1 does not cover *all* functions which admit 2-round protocols. We demonstrate this using the following "degenerate" example, which in fact requires only one round, but more interesting examples (requiring 2 rounds) can be obtained.

Example 1. Consider the function $f(x_1, x_2, \ldots, x_n) = (x_1 \oplus x_2, \perp, \ldots, \perp)$. The value x_i is the input of player P_i, only player P_1 should output the exclusive-or of the bits x_1, x_2 and other players have no output. Note that the output of f depends on inputs of *two* players. Yet, it can be verified that the trivial protocol, in which P_2 sends its input to P_1 and the latter computes the correct output, is t-secure for any threshold t.

In the next section we will show that the above function does *not* admit a 2-round protocol if all players should output $x_1 \oplus x_2$.

4 Negative Results

In this section we prove impossibility of 2-round secure computation for some simple specific functions. Since defining the notion of security is a delicate issue, it is not surprising that negative results may also involve some subtleties. In particular, one has to account for *all* possible strategies of the ideal-process adversary, who is not restricted to any particular behavior pattern. Our general paradigm for proving negative results is the following. For a given function f and a protocol Π, we define a specific real-life adversary $\mathcal{A}_0$ which "breaks" Π in the sense that it has some advantage over any ideal-process adversary $\mathcal{A}'$ attacking π_f. To demonstrate this, we typically define some distribution on the inputs of uncorrupted players, and then specify some concrete "challenge" which no $\mathcal{A}'$

can meet (in the ideal process) as successfully as $\mathcal{A}_0$ by corrupting the same players as $\mathcal{A}_0$ does. For specifying such a challenge, we may use any predicate on the inputs and outputs of uncorrupted players. For instance, $\mathcal{A}_0$ may challenge $\mathcal{A}'$ to guess the input of a specific uncorrupted player, or to fix the output of some uncorrupted player to 0. If we show that $\mathcal{A}_0$ can significantly outperform every $\mathcal{A}'$ in meeting such a challenge, then we have shown Π to be insecure.

4.1 The Functions SB and XOR, or: The Power of Rushing

In this section, we prove negative results for simultaneous broadcast (defined below) and XOR. A common characteristic of these tasks is that they become easier in the fully synchronous model; thus, in addition to proving the necessity of 3 rounds, these examples also serve to separate the fully synchronous model from the standard model.

We start by showing the impossibility of a 2-round simultaneous broadcast (SB) protocol. This natural task is formally defined by the function $\mathrm{SB}(x_1, x_2, \ldots, x_n) = x_1 \circ x_2$, i.e., each player should output the concatenation of the first two inputs. (We refer to each of the two parts of the global output as an *output entry.*) Note that the main security requirement imposed by an SB protocol is *independence*: any ideal-process adversary attacking at most one of the first two players should be unable to induce a non-negligible correlation between the two output entries. We obtain our impossibility result by describing a strategy which allows the adversary to break this independence requirement. The high-level idea is the following. The adversary will corrupt one of the two input holders, say P_2, and some carefully chosen additional player P_j, where $j > 2$. It will pick its Round 1 messages in such a way that will allow its action in Round 2 to have a non-negligible effect on the second output entry (as seen by uncorrupted players). It will then use its rushing capabilities and the fact that P_j was "honest-looking" in Round 1 to first learn the first output entry, and then correlate the second entry with the first one. This will contradict the independence requirement.

We now formalize the above intuition and fill in some missing details. It will be convenient to use the following notation. By (B, M), where $M = (M_1, \ldots, M_n)$, we denote some joint distribution of messages and broadcast sent by P_2 in Round 1 (where M_i is the message sent to P_i). By (B^0, M^0) (resp., (B^1, M^1)) we denote the *honest* distributions corresponding to the input $x_2 = 0$ (resp., $x_2 = 1$). For a distribution (B, M), let $q_\sigma(B, M)$ denote the probability that the protocol's second output entry is equal to 1, given that: (1) $x_1 = \sigma$; (2) P_2's Round 1 messages and broadcast are distributed according to (B, M); (3) everyone else follows the protocol (including P_2 in Round 2). Finally, let $q(B, M) = (q_0(B, M), q_1(B, M))$. All the above distributions and probabilities are parameterized by the security parameter k, which will usually be omitted. We start with the following lemma.

Lemma 2. *The distributions ensembles $(B^0, M_1^0)(k)$ and $(B^1, M_1^1)(k)$ are computationally indistinguishable.*

Proof. Assume towards a contradiction that there is a distinguisher D such that D always outputs 0 or 1, and $|\Pr[D(B^0, M_1^0, k) = 1] - \Pr[D(B^1, M_1^1, k) = 1]| >$

k^{-c}, for some constant c and infinitely many values of k. We use D to show that an adversary corrupting P_1 can break the independence requirement. The adversary's strategy is simple: it waits to hear the broadcast b and message m received from P_2 in Round 1, evaluates $D(b, m, k)$, and uses the result as its input. Clearly, the correlation induced by the adversary cannot be emulated in the ideal process (even up to computational indistinguishability). $\square$

Theorem 2. *There is no 2-round (computationally) secure SB protocol, for any $t \geq 2$ and an arbitrarily large number of players $n \geq 3$.*

Proof. Assume towards a contradiction that a 2-round SB protocol is given. Consider the following four pairs of probabilities:

$$\begin{aligned}
Q_1 &= q(B^1, M_1^1, M_2^1, \ldots, M_n^1) \\
Q_2 &= q(B^1, M_1^1, 0, 0, \ldots, 0) \\
Q_3 &= q(B^0, M_1^0, 0, 0, \ldots, 0) \\
Q_4 &= q(B^0, M_1^0, M_2^0, \ldots, M_n^0)
\end{aligned}$$

By the protocol's correctness, we must have $Q_1 \geq (1 - \mathsf{neg}, 1 - \mathsf{neg})$ and $Q_4 \leq (\mathsf{neg}, \mathsf{neg})$, where neg denotes some negligible function in k. Moreover, by Lemma 2, the difference between Q_2 and Q_3 is negligible. Hence, there is a substantial difference either between Q_1 and Q_2 or between Q_3 and Q_4. Assume wlog that the difference between the first entries of Q_1 and Q_2 (corresponding to the probability q_0) is large, say, more than 1/3. By a hybrid argument, there exists $i \geq 2$ such that $q_0(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0) - q_0(B^1, M_1^1, \ldots, M_{i-1}^1, 0, \ldots, 0) > 1/(3n)$. It follows that one of the two q_0 probabilities above must be different by at least $1/(6n)$ from one of the two corresponding q_1 probabilities. Assume, without loss of generality, that

$$|q_0(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0) - q_1(B^1, M_1^1, \ldots, M_{i-1}^1, 0, \ldots, 0)| > \frac{1}{6n} \qquad (1)$$

(the other cases are similar).

Now, we need to identify two players to complete two tasks in our attack. Yet, it might be the case that both these tasks can be embodied into a single player. We need an honest looking player P_j from whose local view we will compute the correct output, and a second player P_i who can toggle the output of the rest of the players in the protocol. The index of player P_i is given from Eq. (1). If $i > 2$ then this player has acted honestly until now and thus can also be "used" as the player P_j from which we extract the output. Otherwise, i.e. if $i = 2$, then we set $P_j = P_3$, as the player whose view we will examine. An adversary corrupting P_2, P_j can correlate the second output entry with the first as follows. Its Round 1 messages are distributed according to $(B^1, M_1^1, M_2^1, M_3^1, \ldots, M_i^1, 0, \ldots, 0)$. In Round 2, it waits to hear the messages from all uncorrupted players, and then computes the first output entry from the entire view of P_j. Since P_j was honest so far, the protocol's correctness guarantees that it learns the correct output with overwhelming probability.[8] Let α be the value of the first output

[8] Note that there is no guarantee that the correct output can be inferred from the view of P_2, since P_2 has deviated from the protocol in Round 1.

entry computed by P_j. The adversary correlates its output with α by letting P_i, the toggling player, act as follows. If $\alpha = 0$, P_i behaves honestly (i.e., uses the original message M_i^1 received from P_2 in Round 1). Otherwise, it behaves as if this message was set to 0. It follows from Eq. (1) that the second output entry will be significantly correlated with the first, contradicting the independence requirement. □

Note that the SB function admits a trivial 1-round protocol in the fully synchronous model; thus in this case coping with a rushing adversary costs *two* additional rounds. Another observation is that the requirement $t \geq 2$ is essential. Indeed, the following 5-player protocol computes SB with perfect 1-security in two rounds: (1) Each of P_1, P_2 privately sends its input to each of the remaining 3 players; (2) Each of P_3, P_4, P_5 passes the inputs it received to all other players; Each player outputs the majority of the 3 candidates for each input it received in Round 2. It is not hard to verify that the above protocol is a 1-secure SB protocol.

The function XOR. We now turn to the function $\mathrm{XOR}(x_1, x_2, \ldots, x_n)$ defined as $x_1 \oplus x_2$. We show, by refining the previous arguments for the SB function, that this function as well cannot be securely computed in two rounds.

Theorem 3. *There is no 2-round (computationally) secure XOR protocol, for any $t \geq 2$ and an arbitrarily large number of players $n \geq 3$.*

Proof. Similarly to the proof of Theorem 2, the adversary corrupts the player P_2 which, together with an additional player (to be chosen carefully), is used to violate the properties of the alleged protocol. We also follow some of the notations used in the proof of Theorem 2. Specifically, by (B, M) we denote some joint distribution of the broadcast and the private messages (respectively) sent by P_2 in Round 1 of the protocol. By (B^b, M^b) ($b \in \{0, 1\}$) we denote the *honest* distribution corresponding to the input $x_2 = b$. Consider a scenario where P_1 chooses its input x_1, at random. In such a case, in the ideal process, the output is totally random (i.e., each value $\{0, 1\}$ is obtained with probability of exactly 0.5). On the other hand, we will show a strategy for the adversary to significantly bias the output (towards one of the output values). As in the proof of Theorem 2, let $q(B, M)$ be a pair (p_0, p_1) indicating the probability that the output of the function (as seen by the good players) is 1 provided that the input x_1 is 0 or 1 (respectively) and that the first round messages of P_2 are distributed as in (B, M). As before, consider the following four pairs:

$$\begin{aligned} Q_1 &= q(B^0, M_1^0, M_2^0, \ldots, M_n^0) \\ Q_2 &= q(B^0, M_1^0, 0, \ldots, 0) \\ Q_3 &= q(B^1, M_1^1, 0, \ldots, 0) \\ Q_4 &= q(B^1, M_1^1, M_2^1, \ldots, M_n^1) \end{aligned}$$

By the protocol's correctness, it follows that $Q_1 = (\mathsf{neg}, 1 - \mathsf{neg})$ while $Q_4 = (1 - \mathsf{neg}, \mathsf{neg})$. In addition, similarly to Lemma 2, the distributions (B^0, M_1^0) and (B^1, M_1^1) must be indistinguishable (as otherwise P_1, by using its rushing capability in the first round is able to correlate its input x_1 with input x_2 and bias the output; this is impossible to achieve in the ideal process). Hence, the difference between q_2, q_3 is neg. Consider the L_1-distance between two pairs $(p_0, p_1), (p'_0, p'_1)$ (defined as $|p_0 - p'_0| + |p_1 - p'_1|$). It follows that either the distance between q_1 and q_2 is at least $1 - \mathsf{neg}$ or the distance between q_3 and q_4 is at least $1 - \mathsf{neg}$. Assume, without loss of generality, that the first is true. We now argue that, for some i ($2 \leq i < n$), the two pairs

$$(p_0^i, p_1^i) \stackrel{\triangle}{=} q(B^0, M_1^0, \ldots, M_i^0, 0, \ldots, 0)$$

and

$$(p_0^{i-1}, p_1^{i-1}) \stackrel{\triangle}{=} q(B^0, M_1^0, \ldots, M_{i-1}^0, 0, \ldots, 0)$$

are such that either $\max\{p_0^i, p_0^{i-1}\} + \max\{p_1^i, p_1^{i-1}\} > 1 - \mathsf{neg} + 1/(5n)$ or $\min\{p_0^i, p_0^{i-1}\} + \min\{p_1^i, p_1^{i-1}\} < 1 - \mathsf{neg} - 1/(5n)$. Otherwise, this in particular implies that all the points (p_0^i, p_1^i) are such that $1 - \mathsf{neg} - 1/(5n) < p_0^i + p_1^i < 1 - \mathsf{neg} + 1/(rn)$. This in turn implies that the distance between two adjacent pairs is smaller than $1/(2n)$ and the total distance between q_1 and q_2 is less than $0.5 < 1 - \mathsf{neg}$, contradicting what we know about this distance. Suppose that for some i we have $\max\{p_0^i, p_0^{i-1}\} + \max\{p_1^i, p_1^{i-1}\} > 1 - \mathsf{neg} + 1/(5n)$; we describe a strategy for the adversary to bias the output towards 1 (in the other case there is a dual strategy to bias the output towards 0). Now, the adversary picks another corrupted player P_j: if $i > 2$ the adversary uses $P_j = P_i$; otherwise, if $i = 2$ the adversary uses, say, $P_j = P_3$. The adversary lets P_2 play in the first round as in $(B^0, M_1^0, \ldots, M_i^0, 0, \ldots, 0)$. In the second round, the adversary uses P_j as a Trojan horse; it lets P_j first get all the second round messages by other players (rushing) and checks which message from P_2 (either M_i^0 or 0) will cause P_j to output 1 (the idea being that a difference which is only in the first round message sent to P_i will only influence the second round message sent by P_i and no other message). If there is such a message then P_j proceeds as if it got this message (and since P_j is honest-looking its output is the same as the output of all good players); otherwise, the adversary picks one of the two messages arbitrarily. Since x_1 is randomly chosen and by the choice of i, the probability of getting the output 1 is now $0.5 \cdot (\max\{p_0^i, p_0^{i-1}\} + \max\{p_1^i, p_1^{i-1}\})$ which is significantly larger than 0.5, as needed. □

4.2 The Function AND, or: The Advantage of Being Selfish

In this section we consider the function $\text{AND}(x_1, x_2, \ldots, x_n)$ defined as $x_1 \wedge x_2$. We will show that this function cannot be securely computed in two rounds. This case differs from the previous ones in that the relevant impossibility result does *not* rely on the adversary's rushing capability, and thus holds also in the fully synchronous model. The intuition here is also different. In the previous examples, we showed that the adversary could violate the *correctness* of the protocol by inducing some invalid output distribution. In the current case, we will show that the real-life adversary can somehow gain an *information advantage* over its ideal-process counterpart. This will be achieved by practicing a typical *selfish* behavior: the adversary, corrupting P_2 and some other player P_j ($j > 2$), will manage to collect information about x_1 from all uncorrupted players and at the same time refuse to contribute its own share of information to the community. This will allow the real-life adversary to obtain a better prediction of the unknown input than any of the uncorrupted players, which (for the specific case of the AND function) is impossible to achieve in the ideal process.

Theorem 4. *There is no 2-round (computationally) secure AND protocol, for any $t \geq 2$ and an arbitrarily large number of players $n \geq 3$, even in the fully synchronous model.*

Proof. Similarly to the previous proofs, the adversary corrupts player P_2, which together with an additional player is used to violate the properties of the alleged protocol. We also follow some of the previous notation. Specifically, by (B, M) we denote some joint distribution of the broadcast and the private messages (respectively) sent by P_2 in Round 1 of the protocol. By (B^b, M^b) ($b \in \{0, 1\}$) we denote the *honest* distribution corresponding to the input $x_2 = b$. Consider a scenario where P_1 chooses its input x_1 at random. We now argue that in the ideal process the best prediction that the adversary has for the value of x_1 is OUT, the output of the good players, whereas we show a real-life adversary that can guess x_1 with significantly better probability than by using this output. For (B, M) as above, let $\mathsf{COR}(B, M)$ denote the correlation of OUT with x_1 provided that the first round messages by P_2 are distributed as in (B, M). Namely,

$$\mathsf{COR}(B, M) \stackrel{\triangle}{=} |\Pr[\mathrm{OUT} = 1 | x_1 = 1, (B, M)] - \Pr[\mathrm{OUT} = 1 | x_1 = 0, (B, M)]| .$$

Consider the following four quantities:

$$\begin{aligned} q_1 &= \mathsf{COR}(B^0, M_1^0, M_2^0, \ldots, M_n^0) \\ q_2 &= \mathsf{COR}(B^0, M_1^0, 0, \ldots, 0) \\ q_3 &= \mathsf{COR}(B^1, M_1^1, 0, \ldots, 0) \\ q_4 &= \mathsf{COR}(B^1, M_1^1, M_2^1, \ldots, M_n^1) \end{aligned}$$

By the protocol's correctness, it follows that $q_1 = \mathsf{neg}$ while $q_4 = 1 - \mathsf{neg}$. In addition, similarly to Lemma 2, the distributions (B^0, M_1^0) and (B^1, M_1^1) must be indistinguishable (otherwise an adversary corrupting, in a passive manner, the player P_1 gets a significantly better than 50% chance of guessing the input x_2 even when the honest players' output is 0; this is impossible in the ideal process). Hence, the difference between q_2, q_3 is neg. It follows that either the distance between q_1 and q_2 is at least $0.5 - \mathsf{neg}$ or the distance between q_3 and q_4 is at least $0.5 - \mathsf{neg}$. Assume, without loss of generality, that the first is true. Therefore, for some i ($2 \leq i < n$) the quantity $\mathsf{COR}(B^0, M_1^0, \ldots, M_i^0, 0, \ldots, 0)$ is significantly smaller than $\mathsf{COR}(B^0, M_1^0, \ldots, M_{i+1}^0, 0, \ldots, 0)$ (by at least $(1 - \mathsf{neg})/(2n)$). Now, the adversary picks another corrupted player P_j: if $i > 2$ the adversary uses $P_j = P_i$; otherwise, if $i = 2$ the adversary uses, say, $P_j = P_3$. The adversary plays as in the distribution $(B^0, M_1^0, \ldots, M_i^0, 0, \ldots, 0)$[9] so as to guarantee a lower correlation between the output of good players, OUT, and x_1; however, it also uses P_j to compute the output OUT$'$ of the good players (P_j

[9] namely it samples from the distribution (B^0, M^0) and creates the hybrid distribution by replacing some of the messages by 0-messages.

behaves like such a player) in case the messages of P_2 come from the distribution $(B^0, M_1^0, \ldots, M_{i+1}^0, 0, \ldots, 0)$. This value OUT$'$ is significantly better correlated with x_1 than the actual output OUT.

Finally, we argue that in the ideal process the adversary has no better predictor for the value of x_1 than OUT. For this, simply consider all the 4 potential views (x_2, OUT) that the adversary may see. The view $(0, 1)$ is impossible; for both views $(1, 0)$ and $(1, 1)$ we have $x_1 = \mathrm{OUT}$. If the view is $(0, 0)$ then the adversary has no information about x_1; in such a case, guessing the value $x_1 = \mathrm{OUT}$ is correct with probability $1/2$ and is as good as any other way of guessing. Hence, OUT is an optimal predictor for x_1. □

5 Concluding Remarks

We have answered some of the main qualitative questions concerning the round complexity of secure multiparty computation in our standard model. In particular, we have shown that security against an active adversary requires strictly more interaction than security against a passive adversary, and that general secure computation tasks require more interaction than distributed zero-knowledge and similar tasks. As a future goal, it remains to find a characterization of secure computation tasks according to their exact round complexity. This question appears to be nontrivial, partly due to the difficulty of capturing the exact power of an adversary attacking an ideal-process implementation of complex functions.

References

1. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proc. 8th ACM PODC*, pages 201–209. ACM, 1989.
2. D. Beaver. Minimal-Latency Secure Function Evaluation. In *Eurocrypt '00*, pages 335–350, 2000. LNCS No. 1807.
3. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead (extended abstract). In *Proc. of CRYPTO '90.*
4. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. 22nd STOC*, pages 503–513. ACM, 1990.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. *Proc. 20th STOC88*, pp. 1–10.
6. M. Ben-Or and N. Linial. Collective Coin-Flipping. In *Randomness and Computation*, pages 91–115, 1990.
7. P. Berman, J. Garay, and K. Perry. Bit Optimal Distributed Consensus. In R. Yaeza-Bates and U. Manber, editors, *Computer Science Research*, pages 313–322. Plenum Publishing Corporation, 1992.
8. C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. In *Proceedings of ICALP'00*, 2000.
9. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
10. D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *Proc. 20th STOC88*, pages 11–19.

11. R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Proc. Crypto 2001.*
12. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations with dishonest minority. In *Eurocrypt '99*, pages 311–326, 1999. LNCS No. 1592.
13. Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proc. 26th STOC*, pages 554–563. ACM, 1994.
14. P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. *SIAM. J. Computing*, 26(2):873–933, 1997.
15. F. Gavril. Manuscript, 1974.
16. R. Gennaro, S. Halevi, and T. Rabin. Round-optimal zero knowledge with distributed verifiers. www.research.ibm.com/security, 2002.
17. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *Proc. 33th STOC*. ACM, 2001.
18. Rosario Gennaro. Achieving Independence Efficiently and Securely. In *Proc. 14th ACM PODC*, pages 130–136. ACM, 1995.
19. O.Goldreich. Secure multi-party computation (manuscript). www.wisdom.weizmann.ac.il/∼oded/pp.html, 1998.
20. O. Goldreich. *Foundation of Cryptography – Fragments of a Book*. ECCC 1995. Available online from *http://www.eccc.uni-trier.de/eccc/*.
21. O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *Proc. 19th STOC*, pages 218–229. ACM, 1987.
22. Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS97*, pages 174–184, 1997.
23. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st FOCS*, 2000.
24. Y. Ishai and E. Kushilevitz. Perfect Constant-Round Secure Computation via Perfect Randomizing Polynomials. In *Proc. ICALP '02*.
25. L. Lamport, R.E. Shostack, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. and Systems*, 4(3):382–401, 1982.
26. Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In *Crypto '01*, pages 171–189, 2001. LNCS No. 2139.
27. N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.
28. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM PODC*, pages 51–59. ACM, 1991.
29. M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.
30. T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proc. 21st STOC*, pages 73–85. ACM, 1989.
31. T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing For NC1. In *Proc. 40th FOCS*, pages 554–567. IEEE, 1999.
32. A. C-C. Yao. How to Generate and Exchange Secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.